View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Universidade do Minho: RepositoriUM

Do-It-Yourself Shared-Storage Cluster with an Off-the-Shelf DBMS*

Luís Soares
University of Minho
los@di.uminho.pt

José Pereira
University of Minho
jop@di.uminho.pt

1. Motivation

Database server clustering, used ubiquitously for dependability and scalability, is commonly based on one of two competing approaches. The first, a *shared storage* cluster such as Oracle RAC, is based on distributed locking and a distributed cache invalidation protocol [3]. Its main advantage is flexibility, as one uses as many nodes as required for processing the workload and to ensure the desired availability, while the storage is configured solely according to the desired storage bandwidth and disk resilience. Unfortunately, it can be implemented only with a deep refactoring of server software; the reliance on distributed locking limits its scalability; and by directly sharing a physical copy of data it becomes harder to ensure data integrity. These reasons make it costly to develop and deploy, as attested by most of the mainstream database servers not providing this option.

The second, a *shared nothing* cluster such as C-JDBC, can be implemented strictly at the middleware level by intercepting client requests and propagating updates to all replicas [1]. The resulting performance and scalability is good, especially, with currently common mostly read-only workloads. Moreover, as each replica is physically independent, this strategy can easily cope with a wider range of faults [2]. The main problem is that in a shared-nothing cluster a separate physical copy of data is required for each node. Therefore, even if a only few copies are required for dependability, a large cluster with hundreds of nodes must be configured also with sufficient storage capacity for hundreds of copies of data.

The naive combination of both approaches by simply sharing the data volume would obviously not work, as asynchronous propagation of updates and non-deterministic physical data layout would lead to data inconsistency and ultimately to corruption. Simultaneously obtaining the advantages of both approaches is however appealing.

2. Approach

Our approach [5] achieves such goal, of running shared nothing server software on a shared storage cluster and combining the advantages of both, by adding a system level I/O interceptor layer as shown in Fig. 1. This layer intercepts all file I/O operations issued by the DBMS server and provides it the abstraction of multiple non-shared physical copies as described below. In detail, one of the nodes (1), the *writer*, is allowed to write back to shared-storage. Other nodes (2), the *copiers*, perform copy-on-write locally to volatile storage (3), *e.g.*, a RAM disk, when necessary to ensure that future read operations will not observe pages modified elsewhere.

During normal operation, all nodes try to update the shared volume as they commit replicated updates. One makes sure that updates by each copier are not visible elsewhere by storing a local copy of the page in volatile storage, and using it for subsequent reads. Updates by the writer can only proceed after ensuring that the page is stable, *i.e.*, that all other nodes, the copiers,

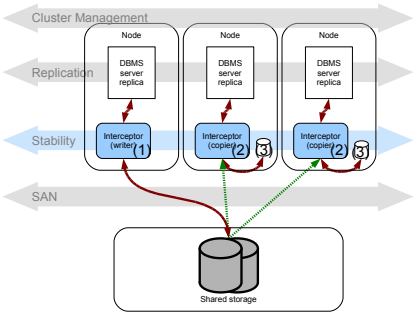

Figure 1. Proposed architecture.

*This work was partially supported by project "PASTRAMY: Persistent and highly Available Software TRansactional MemorY" (PTDC/EIA/72405/2006).

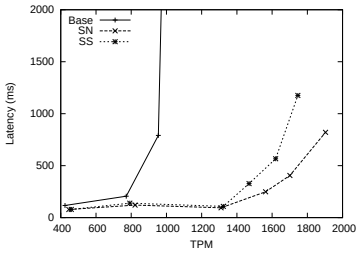

Figure 2. Throughput and latency results.

have a local copy of that page. Stability is ensured by the following protocol: When the writer node issues a write request, it is blocked and all nodes requested to fetch the previous value of the data from shared storage and store it as a local copy. Upon receiving replies from all nodes, the write request can proceed. Read and sync requests are not intercepted at the writer. This needs to be done only once for each block, thus reducing the number of distributed interactions required. Write requests from copier nodes are directly routed to the copy without any distributed interaction whatsoever. Read requests are first served from the local copy and if unavailable, because yet unwritten, from the shared storage. Sync requests are ignored, since there is no persistent storage involved.

To prevent volatile local copies of becoming increasingly large, as pages are updated by different nodes and copies need to be created, each copier is periodically restarted, thus flushing its local cache and restarting from the shared copy maintained by the writer node. The net result is a shared-storage cluster with minimal changes to existing DBMS software, where a single copy of the data, indistinguishable from what a single DBMS instance would create, is shared.

3. Discussion

To evaluate the feasibility of the proposed approach, we have implemented it on MySQL Server 5.1 by intercepting all file I/O system calls issued by the InnoDB subsystem. Then we compared the resulting shared-storage configuration (SS) with two configurations that can be achieved with the original server software: a shared-nothing configuration (SN) and a baseline configuration using a single server (Base). All tests were performed on the same 2 server cluster hardware and used the same workload generated according to the TPC-W benchmark *ordering mix*. In detail, read-only transactions are evenly distributed among available servers while update transactions are executed by all of them.

The results are shown in Fig. 2, showing aggregate throughput and average latency as load is increased by adding emulated clients. Namely, the Base configuration peaks at approximately 800 TPM while, as expected in such a small cluster, the SN configuration achieves roughly linear scalability, doubling the maximum achievable throughput. Most interestingly, our SS configuration, using the same storage resources as Base, provides performance comparable to SN which must use twice the storage space and bandwidth. This means that the SS configuration can be configured with exactly as much storage redundancy as desired, exactly as Base and other shared-storage approaches, in contrast to the SN configuration in which one must use at least as many physical disk copies as cluster nodes.

Current work focuses on experimental evaluation [4] to determine the overhead of the stability and garbage collection mechanisms. The current implementation based on MySQL 5.1 is available at `http://holeycow.org`. Preliminary results obtained indicate that indeed the overhead of the stability protocol is negligible. By measuring the growth of local copies, one can also conclude that garbage collection by periodical restarts should be feasible.

Finally, the containment of each replica by its local copy-on-write file should also provide important resilience guarantees in face of faults, such as software bugs in the DBMS server, that lead to memory and disk corruption. Moreover, the limited dependency on shared structures introduced by our approach should also be advantageous to enable on-line upgrade of the DBMS server to a new version.

References

[1] E. Cecchet, J. Marguerite, and W. Zwaenepoel. Partial replication: Achieving scalability in redundant arrays of inexpensive databases. In *Intl Conf. Principles of Distributed Systems (OPODIS)*, 2003.

[2] I. Gashi, P. Popov, and L. Strigini. Fault tolerance via diversity for off-the-shelf products: A study with sql database servers. *IEEE Tran. Dependable and Secure Computing*, 4(4):280–294, Oct.-Dec. 2007.

[3] T. Lahiri, V. Srihari, W. Chan, N. Macnaughton, and S. Chandrasekaran. Cache fusion: Extending shared-disk clusters with shared caches. In *Intl. Conf. Very Large Databases (VLDB)*, 2001.

[4] Luís Soares and José Pereira. Implementation and evaluation of the HoleyCoW proof-of-concept. http://holeycow.org, 2009.

[5] Luís Soares and José Pereira. A simple approach to shared storage database servers. In *Proc. of 3rd Ws. Depdendable Distributed Data Management (with EuroSys'09)*, 2009.